6-1-2017

NovenaNetwork: A Catholic Social Media iOS Application for Praying Novenas as a Community

Marissa Le Coz
Dartmouth College

NovenaNetwork: A Catholic Social Media iOS Application for Praying Novenas as a Community

Marissa Le Coz

Computer Science Technical Report TR2017-833
Dartmouth College

June 2017

Contact: Marissa.C.Le.Coz.GR@dartmouth.edu

NovenaNetwork: A Catholic Social Media iOS Application for Praying Novenas as a Community

Marissa Le Coz '17
Dartmouth College
COSC 99

Advisors: Xing-Dong Yang & Sean Smith

Abstract

Recent years have seen the rise of a Catholic presence on the Internet and on social media in particular. At the same time, there has been a movement among young Catholics to rediscover and revitalize the traditions of the Church. Novenas, an ancient form of spirituality involving praying for nine consecutive days for some special purpose, are one such tradition. NovenaNetwork, which debuted in the App Store on May 23, 2017 and acquired over 50 users in three countries within its first six days, is an iOS social media application for praying novenas with others. While multiple apps and websites already exist for praying novenas, NovenaNetwork is the first social media network on any platform for doing so, making it unique in this space. I will begin by presenting the functionality of NovenaNetwork in depth and examining how the app differs from its competitors. Next, I will outline the design and implementation of the app. Finally, I will offer an evaluation of the app and unveil features I intend to implement in subsequent releases of NovenaNetwork.

Table of Contents

1. Introduction

Novenas (from the Latin ***novem***, meaning "nine") are a traditional form of Catholic prayer dating back to the earliest days of the Church.[1] Novenas involve praying for nine consecutive days for some particular prayer ***intention*** (i.e., thing for which one is praying). Some novenas are prayed to God, while others are prayed to some saint, asking him/her to pray to God on the supplicant's behalf (i.e., to ***intercede*** for the supplicant). As with any form of prayer, the point of novenas is not for the supplicant to get what s/he wants. Rather, the purpose of praying novenas is to grow closer to God, to foster regular prayer habits, and to learn to better trust in the will of God.

According to Biblical accounts of the early Church, just before Jesus ascended to Heaven, He told his disciples to stay in Jerusalem until they were baptized with the Holy Spirit.[2] The disciples prayed together "constantly" as they waited in Jerusalem.[3] After nine days of prayer,[4] on the feast of Pentecost, the disciples received the gift of the Holy Spirit.[5] The practice of praying novenas is derived from Jesus's first disciples' nine days of prayer preceding the reception of the Holy Spirit.

Throughout the history of the Church, novenas have taken on many forms. Early Christians would pray for nine days for the souls of recently deceased loved ones. In the Middle Ages, it was common practice to pray a novena for the nine days leading up to Christmas, signifying the nine months that Jesus spent in His mother Mary's womb. Particularly before the Second Vatican Council (1962-1965), many Catholic parishes (i.e., individual churches) would pray novenas for various intentions as a community.[6] Today, traditional Catholic practices (of which novenas are a part) have been making a resurgence, especially among young adults. As one Catholic priest, Father Joseph Kramer, has put it in an interview with ***USA Today***, "there is a movement among young Catholics to know, discover, and preserve their Catholic heritage."[7]

Despite statistical report after statistical report showing the number of Catholics in the United States declining,[8] it is important to note that there still exists a vital and vibrant Catholic subculture in America (and around the world). As evidence of this, consider the social media following of some popular online Catholic ministries. ChurchPop.com, which is essentially a Catholic version of Buzzfeed, has 334,833 likes on Facebook at the time of writing this paper. Catholic Memes, a Facebook page whose name is self-explanatory, has 316,308 likes. Catholic Answers, a non-profit that provides resources for intellectually understanding the Catholic faith, boasts 589,006 likes on Facebook. One wildly popular priest who makes videos for YouTube, Bishop Robert Barron, has 106,112 subscribers to his channel, and Father Mike Schmitz, chaplain at the University of Minnesota at Duluth, has 56,268. This evidence indicates that there is a non-negligible number of technologically savvy Catholics on social media who are interested enough in their faith to integrate it into their Internet habits. This niche audience serves as the target user group of my app, NovenaNetwork.

NovenaNetwork is an iOS social media application for praying novenas with others. Users have the opportunity to ***post*** novenas for particular prayer intentions. These posts show up in a ***World Feed***, which is viewable by any user of the app. Other users may then opt to ***join*** these public novenas. Everyone who has joined prays on the same nine days for the same intention, with the start date specified by the creator of that novena post. Every individual user chooses a time for daily push notification reminders to

pray. The next section of this paper will discuss NovenaNetwork's functionality in greater detail. This paper will then turn to exploring the main competitors in this space, highlighting how NovenaNetwork is different. This paper will also outline the iterative design process that led up to the implementation of the app, as well as explain key aspects of this implementation. Finally, this paper will conclude with an evaluation of version 1.0 of NovenaNetwork, as well as unveil future plans for improvements to the app.

2. Exposition of the System

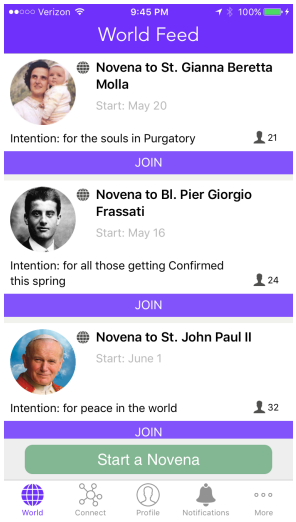

Figure 1. World Feed

When users launch NovenaNetwork, they are greeted with a loading screen, which soon transitions to one of two screens: the first-time user screen or the World Feed screen. In the former case, the user is prompted to create a username. After the user's proposed username is verified as not being already taken, the new user is also directed to the World Feed screen.

To avoid ambiguity of language, from here on out, we will make a distinction between the terms "novena" and "novena post" as it pertains to this system. A ***novena*** shall be defined as one of the particular novenas (i.e., set of prayers) pre-provided by NovenaNetwork, such as the Novena to the Holy Spirit or the Novena to Saint Monica. A ***novena post*** shall be defined as a user-initiated instance of a novena, which includes a prayer intention, start date, and selected novena.

a. World Feed and Joining/Leaving Novena Posts

The World Feed displays all of the novena posts that users of NovenaNetwork have initiated. Each post includes the novena being prayed, an associated image, the start date, the intention, and the number of people who have joined that novena post. To the left of the novena name is a small globe icon, indicating that this novena is visible to all users of NovenaNetwork. (This, at first glance, appears extraneous given that these posts are on the ***World*** Feed. We will see later, however, that "visibility" icons are needed elsewhere in the app; hence, they appear on the World Feed, as well, for the sake of consistency.) At the bottom of each post is a purple button that reads either "DETAILS" or "JOIN." The default for any post is "JOIN" until a) the start date of the novena post has passed, or b) the user whose account the World Feed is being viewed on has already joined the novena post.

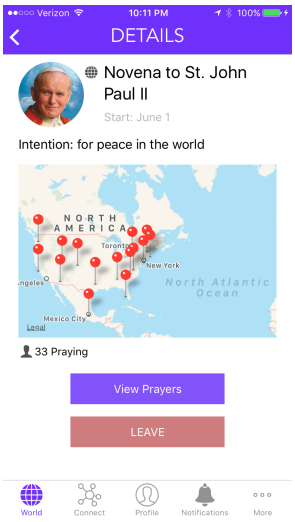

Figure 2. Additional details for a novena post

Tapping on a post takes users to a page with more information on that novena post. This page again includes the novena being prayed, an associated image, the start date, the intention, the number of people who have joined this novena post, and a "visibility" icon. Additionally, this page displays a map of the world with pins at the GPS locations of all those who

have joined this novena post. This page also includes a subset of four possible buttons (with at most two displayed). First, there is always a "View Prayers" button, which sends the user to another page containing the prayers for the associated novena. The second button, if displayed at all, may serve one of three functions. First, it can be a "JOIN" button if three conditions are met: 1) the user did not initiate this novena post, 2) the user has not joined this novena post, and 3) the novena post's start date has not yet passed. Second, it can be a "LEAVE" button when three conditions are met: 1) the user did not initiate this novena post, 2) the user has joined this novena post, and 3) the novena post's start date has not yet passed. Finally, it can be a "DELETE" button if two conditions are met: 1) the user initiated this novena post, and 2) no one else has joined the novena post yet.

If the user taps the "JOIN" button, s/he is taken to a page where s/he can choose a time for daily push notification reminders to pray. When the user selects a time, the system makes the appropriate changes to the database while an activity indicator is displayed to inform the user that things are going on behind the scenes. The user is then brought back to the novena post page with the map. In the UI, the user's coordinate has been added to the map and the number of people praying has been augmented. (This UI update operates separately from the corresponding database update so that the user can get the timeliest visual response possible.) If the user taps the "LEAVE" button, the appropriate changes are made in the database while an activity indicator lets the user know that the system is working. In the UI, the user's GPS coordinate is removed from the map and the number of people praying is decremented. When this is complete, the button transforms to a "JOIN" button. Finally, if a user is presented with the "DELETE" button and taps that, the novena post is deleted from the database, and the user is brought back to the World Feed. As error prevention, none of these buttons can be double tapped. (Each is disabled right after it is tapped so that the same database operations are not invoked twice in succession, which would get particularly messy since these operations are asynchronous.)

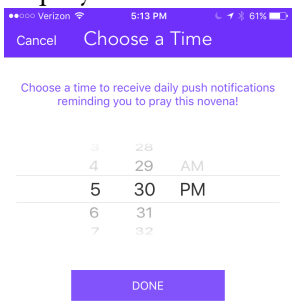

Figure 3. Choosing a time for push notifications when joining a novena post

b. Creating a Novena Post

At the bottom of the World Feed screen, there is a green "Start a Novena" button. Tapping this walks the user through a three or four step process for initiating a novena post. In Step 1, users choose a novena from a collection of twenty novenas. (In future releases of this app, I would like to expand this library.) If users wish to view more information about a novena, they can tap the info icon corresponding to that novena. Suppose the user tapped on the info icon for the Novena to St. Gianna Beretta Molla. The user would be taken to a page with a picture of St. Gianna, a little bit of information about her, and buttons leading to each of the nine days' prayers.

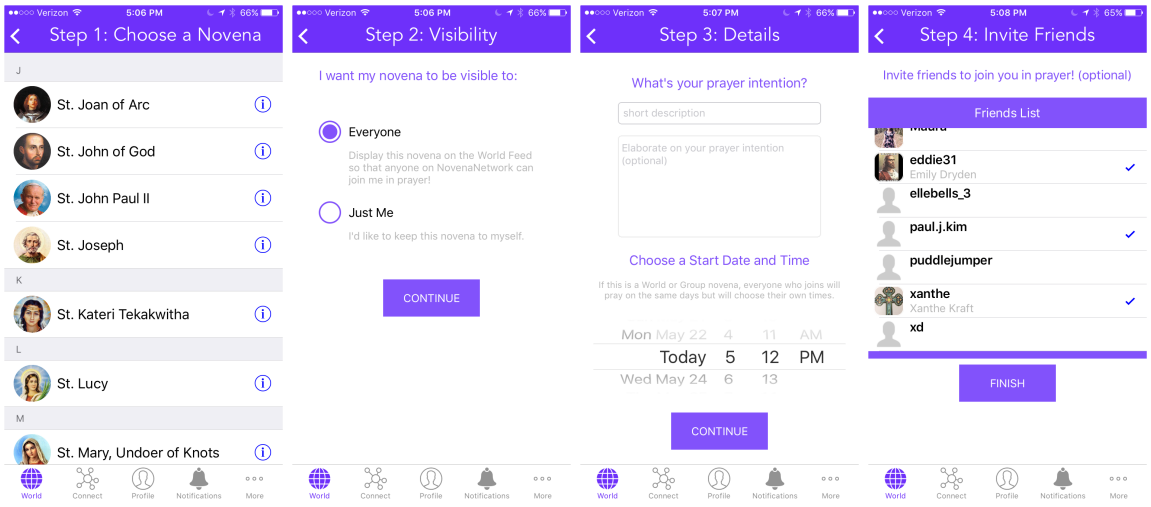

Figure 4. Process for creating a novena post

Once a user selects a novena, s/he must complete Step 2 of the novena post creation process. This step pertains to visibility. Users have the option to make their novena post visible to all users of the app (by displaying it on the World Feed), or to initiate a private novena that only they can see.

Step 3 of the process involves providing other miscellaneous details for the novena post. First, the user must provide a short description of his/her prayer intention (the length of which is enforced by a character limit). This field is required, and the user will not be allowed to proceed until this field is filled out. Second, the user has the option to elaborate on his/her prayer intention. (This elaboration would be displayed on the page for the novena post with the map.) Finally, the user selects a start date for the novena post, as well as a time for daily push notification reminders to pray. The user is prevented from choosing past dates (and is also prevented from choosing a start date more than one year into the future). All users who join this novena post pray on the same nine days, but each person selects his/her own time for push notifications.

If the user is creating a private novena post, Step 3 is the final step in this process. If, however, users are creating a public novena for display on the World Feed, they go to Step 4, where they can invite friends on NovenaNetwork to join them in prayer. After users select the friends from their Friends List that they want to invite and tap the "FINISH" button, the new novena post is created, and each invited friend receives a message in their Notifications tab of the app (discussed later).

c. Viewing Novena Posts Initiated or Joined

NovenaNetwork is a "tabbed" application, meaning that it has tabs along the bottom of the screen that give the user access to different parts of the app. So far, all of the functionality we have looked at has derived from the "World" tab of the app, which corresponds with the World Feed. If users wish to view all of the novena posts that they have initiated and joined, they can visit the "Profile" tab. This screen resembles the World Feed screen in that it consists of novena posts. However, the posts are split into two sections: "Novenas Initiated" and "Novenas Joined." The Profile page is where the "visibility" icons discussed above become important. The globe icon appears on public

novena posts, whereas a lock icon appears on private novena posts. Users can tap on novena posts in these sections to view the more detailed page for that novena post (with the map).

On the Profile page, users may also edit their "real name" and profile picture (by selecting an image from their device's camera roll). Finally, users also have access to the "Start a Novena" button on their profile, which initiates the three or four step process for creating a private or public novena post.

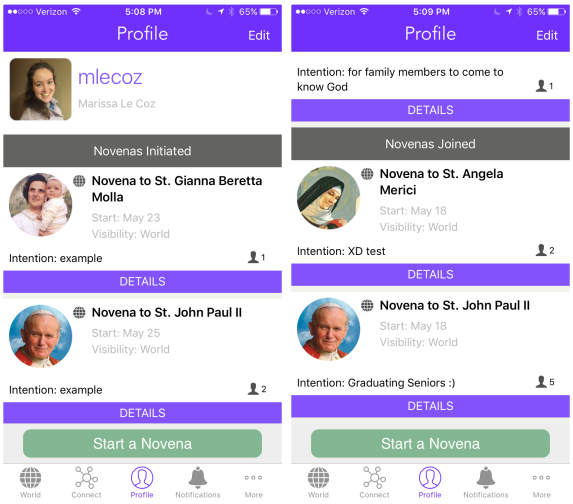

Figure 5. User profile

d. Connecting with Others on NovenaNetwork

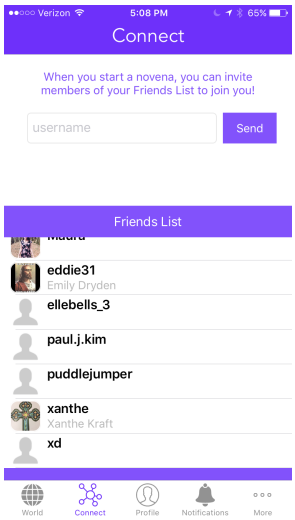

Figure 6. Connect tab used for "friending"

While NovenaNetwork is indeed a social network, it does not include all of the elements of typical social networks; this choice was intentional and will be discussed further in the "Design" section of this paper. As a result of this decision, on NovenaNetwork, "friending" others serves only one purpose: it enables users to send notifications to their friends inviting them to pray as part of Step 4 in the novena post creation process, discussed above.

The "Connect" tab consists of a "Friends List" and a search bar for finding friends by their username. The user search accounts for many edge cases, such as a) the case where the friend requester is already friends with the requestee, b) the case where the friend requester has already sent a friend request to the requestee, but the requestee has not yet accepted or rejected it, c) the case where the requester enters an invalid username, d) the case where the ***requestee*** has already sent the requester a friend request, among other edge cases. Users may unfriend members of their Friends List by swiping left (revealing a "Delete" button) on the username of that person. (This uses iOS's familiar paradigm for deletion.)

e. Viewing Notifications

The "Notifications" tab of NovenaNetwork displays three kinds of notifications: novena post invites, novena post joins, and friend requests. Each notification displays a timestamp, associated image, and message. Notifications have a purple "unread" dot until they have been tapped.

(1) Novena post invites. When users initiate a public novena, in Step 4 of the creation process, they have the option to invite friends from their Friends List to join them in prayer. Invited friends receive a notification of this invitation. Tapping on a novena post invite notification takes the user to the detailed page for that novena post (with the map and "JOIN" button). (If the novena post has been deleted by its creator, a message pops up telling the user that this is the case.)

(2) Novena post joins. When "milestone" numbers of people join a novena post the user has initiated, the user receives notifications. These milestone numbers are five and any multiple of ten. (These numbers do not include the initiator of the novena post.) Tapping on a novena post join notification takes the user to the detailed page for that novena post (with the map).

(3) Friend requests. Friend requests arrive as notifications. Tapping on a friend request notification takes the user to a page with the friend requester's username, profile picture, and real name (if s/he has given it). The recipient of the request has three options: tap the "Accept" button, tap the "Delete" button, or tap the "Cancel" button so as not to respond. If "Accept" or "Delete" is tapped, the appropriate changes are made to the database and the friend request notification disappears. If "Cancel" is tapped, the notification remains so that the user can respond at another time if s/he chooses.

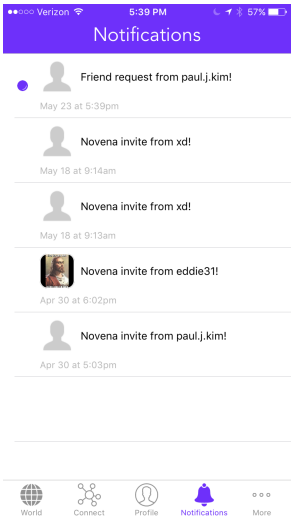

Figure 7. Notifications tab

f. Viewing Prayers and Novena Information

Each novena has a corresponding informational page displaying a related image, a description of the saint to whom the novena is directed (if applicable), and the prayers for each of the nine days. This informational page is accessible in two ways. First, this is the page linked to by the "View Prayers" button on the detailed page for a novena post, as alluded to above. Second, this page is accessible via the info icons in Step 1 of the novena post creation process, also mentioned above.

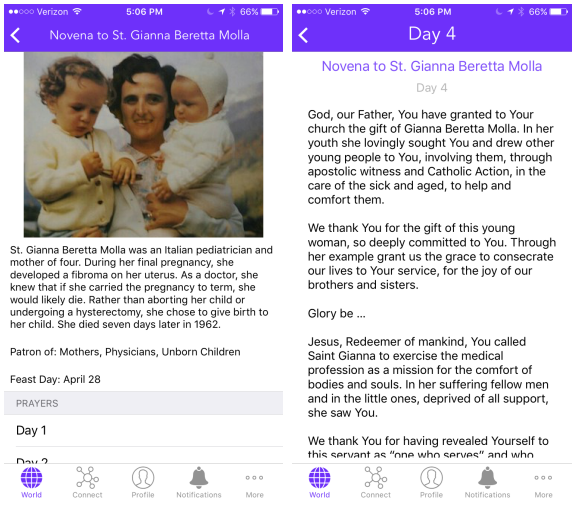

Figure 8. Novena information (left) and an example of a prayer accessible from the novena information page (right)

g. Push Notifications

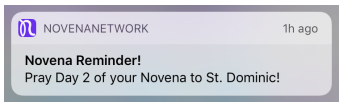

Figure 9. Push notification from NovenaNetwork

As discussed above, when users initiate or join novenas, they select a time for push notification reminders to pray. When a user taps "Finish" (in the case of initiating a novena post) or "Done" (in the case of joining a novena post), all nine push notifications are scheduled. If the user deletes or leaves a novena post, the corresponding push notifications are deleted. Prayer reminder push notifications include the name of the novena as well as which of the nine days the participant is on. The prayer intention is not included in case it is of a personal nature. Push notifications are an important design requirement for an app for praying novenas; remembering to pray a novena for nine consecutive days is a challenge even for the most organized people.

h. About and Attributions

The "More" tab includes an "About" page and an "Attributions" page. The "About" page provides information about the practice of praying novenas, as well as information about NovenaNetwork. The Attributions page provides a URL containing detailed citation information for all images, prayers, icons, etc. used by NovenaNetwork.

3. Competitors

The first and best-established competitor in this space is ***PrayMoreNovenas.com***, which, at the time of writing this paper, has 134,741 likes on Facebook. Like NovenaNetwork, this service is free to use. Its model, however, is different from that of NovenaNetwork. The administrators of PrayMoreNovenas.com choose which novenas to pray and when (some novenas have traditional start dates). Users of the website may then sign up for these novenas and receive daily email reminders to pray. NovenaNetwork is different from PrayMoreNovenas.com in four main ways. First, NovenaNetwork is for the iOS platform, whereas PrayMoreNovenas.com is for the web platform. Second, NovenaNetwork is social media, whereas PrayMoreNovenas.com is not. Third, users of PrayMoreNovenas.com do not initiate novenas, whereas on NovenaNetwork, they do. Finally, users of PrayMoreNovenas.com cannot customize the times of day at which they receive email reminders, whereas users may specify their own times for reminders on NovenaNetwork.

The second real competitor for NovenaNetwork is another iOS app called ***Pray: The Catholic Novena App*** (which will simply be referred to as "Pray" from here on out). Pray is a nicely designed app, but again, its model is different from that of NovenaNetwork. Pray has a library of twenty-three novenas (at the time of writing this paper), but only fourteen are available without making an in-app purchase. NovenaNetwork has a library of twenty novenas, all of which are free to access. The biggest difference between NovenaNetwork and Pray is that Pray is not social media, whereas NovenaNetwork is. Pray encourages the communal praying of novenas—the app makes it easy for users to share on Facebook that they are praying. The idea is for users to invite their Facebook friends to download the app and pray with them in parallel. There is no way to see the number of people who have decided to pray with you as there is on

NovenaNetwork. While Pray encourages users to ***utilize*** social media to get people to pray with them, Pray itself is not a social media app.

4. Design

a. Design Requirements

From my knowledge of Catholicism and Catholic culture, as well as from observing what the competitors in this space have done, I determined four main design requirements (although I did not recognize them as "design requirements" at the time since I had not yet taken COSC 67 (Introduction to Human-Computer Interaction)).

(1) ***Reminders to pray novenas.*** Remembering to do anything for nine consecutive days is hard. The big selling point of PrayMoreNovenas.com and Pray is that they remind users to pray via email or push notifications. In fact, PrayMoreNovenas.com dubs itself "The Original Novena Reminder."

(2) ***A way to pray with/for others.*** Communal prayer and praying on behalf of others is a big deal in Christianity. At every Catholic Mass, the "Prayers of the Faithful" are read, during which the congregation prays for special intentions written up for that week (examples might include praying for Pope Francis, or praying for victims of a terrorism attack, or praying for virtues like humility). Further, most Catholic churches have a "Book of Petitions" where churchgoers anonymously write down intentions for which they are praying. Sometimes these intentions are mentioned at Mass during the Prayers of the Faithful; alternatively, parishioners may peruse the Book of Petitions and join in praying for those intentions. On a more personal level, it is not atypical for one friend to ask another friend to pray for him/her, or for one friend to offer to pray for another. All of this is to say that the notion of intercessory prayer (whether by the saints in Heaven or by living friends) is deeply engrained in Catholic culture and theology.

(3) ***A sleek, modern look.*** The Catholic Church is not known for its technological prowess. When I was perhaps fourteen or fifteen years old, the poor technological quality of my parish's website contributed to my belief that the Catholic Church was (technologically and theologically) old, antiquated, and dying. My views on Catholicism have obviously changed a lot since then, but what I have learned from personal experience is that people can judge organizations based on the quality of their technology (even if that judgment is subconscious). The Catholic Church is in need of more apps and websites that are beautifully designed.

(4) ***A selective use of social media paradigms.*** While social media like Facebook and Twitter can be used well, they can also be used poorly. There was a point during my prototype iterations when I introduced ***too*** many social media paradigms into NovenaNetwork. Something about this felt wrong, but I could not figure out exactly what. I discussed my discomfort with the campus minister at the Aquinas House Catholic Student Center, Meg Costantini, who holds an M.Div. from Notre Dame. As we talked it out, I came to understand that I did not want NovenaNetwork to primarily be a social media app. Rather, I wanted NovenaNetwork to be primarily about praying with others, selectively borrowing from common social media features to achieve this

end goal. For example, it is not necessary to display who posted a novena post on the World Feed. The focus is on praying together (and knowing that you are praying with others), not ***who*** specifically is praying. Similarly, it is not necessary to know who has joined each novena post. The concept of a publically viewable "profile page" did not make sense either. The most logical information to put on such a page would be the novena posts the user has started and joined. Making it known who is praying which novenas makes for an uncomfortable situation—it invites the judgment of other users of NovenaNetwork. ("Wow, he prays a lot of novenas!" or "Hmm, she's slacking in her prayer life!") It also creates an odd sort of peer pressure that makes the motivation to pray a little less genuine. As Jesus said in Matthew 6:6, "Whenever you pray, go into your room and shut the door and pray to your Father who is in secret; and your Father who sees in secret will reward you."[9] Embracing social media paradigms in their fullness in the context of NovenaNetwork seemed to make things public that should not be public. NovenaNetwork in its final form thwarts certain typical social media standards; while this may initially confuse the user, I maintain that it was ultimately the right decision to make.

b. Iterative Design Process

I first came up with NovenaNetwork's concept in November 2015. I originally planned to implement it during the December vacation of my junior year, but I quickly realized a) that this was not the sort of project I could complete in one month, and b) that I did not know enough about Computer Science yet to even know where to start, having only been coding for roughly a year and a half at that point. I placed my vision on a metaphorical shelf until my senior summer, when I had an internship in cloud computing. By the end of that internship, I started thinking about NovenaNetwork again and how I could apply what I had learned that summer about Amazon AWS (a cloud service) to NovenaNetwork. (I ultimately ended up using Apple's CloudKit instead of Amazon's AWS for financial reasons, but my knowledge of AWS definitely helped me to wrap my mind around using CloudKit.) It was then that I decided that I wanted to make NovenaNetwork for my senior culminating experience, if at all possible. In this section, I will walk through my iterative design process for this app.

In 2015, I mapped out some early versions of the app's UX, and I made an extremely ugly interactive prototype (which I thought was fairly sophisticated at the time) using a free interactive prototyping tool called JustInMind. A year and a half later, I used this prototype to explain my project idea to potential advisors. Below, you will see an early UX sketch and a screenshot of the main feed of this very primitive prototype.

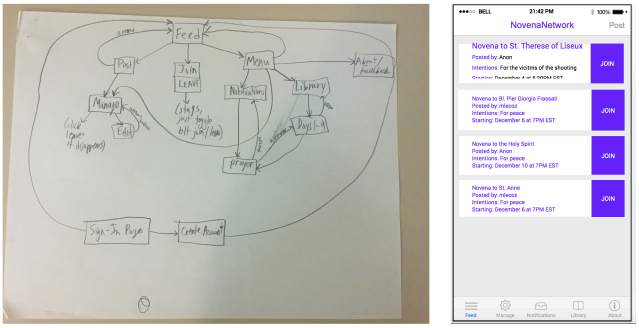

Figure 10. Early sketch and prototype from 2015

A few days before my first term of COSC 99 began, I did some sketching—I had some new ideas for refining the system. On December 28, 2016, I re-mapped the UX for the app. On that particular day, I was concerned that my app idea was too similar to the ones already out there (even though it really is not). So in order to distinguish my app from other technologies for praying novenas, I toyed with the idea of allowing users to "start" a Rosary (another form of Catholic prayer) for particular prayer intentions, such that other users would sign up to pray a Rosary for that intention. The plan was to have this option in addition to starting novena posts so that the focus would be more on social media for praying and less on novenas in particular. You can see that sketch on the right.

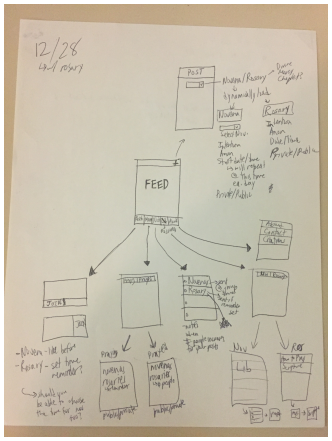

Figure 11. Sketch of app flow from December 28, 2016

A few days later, on December 31, 2016, I decided that adding the Rosary made the app too unfocused, so I revised my UX sketch for the app and made detailed corresponding UI sketches with crayons. Below you can see a subset of these sketches.

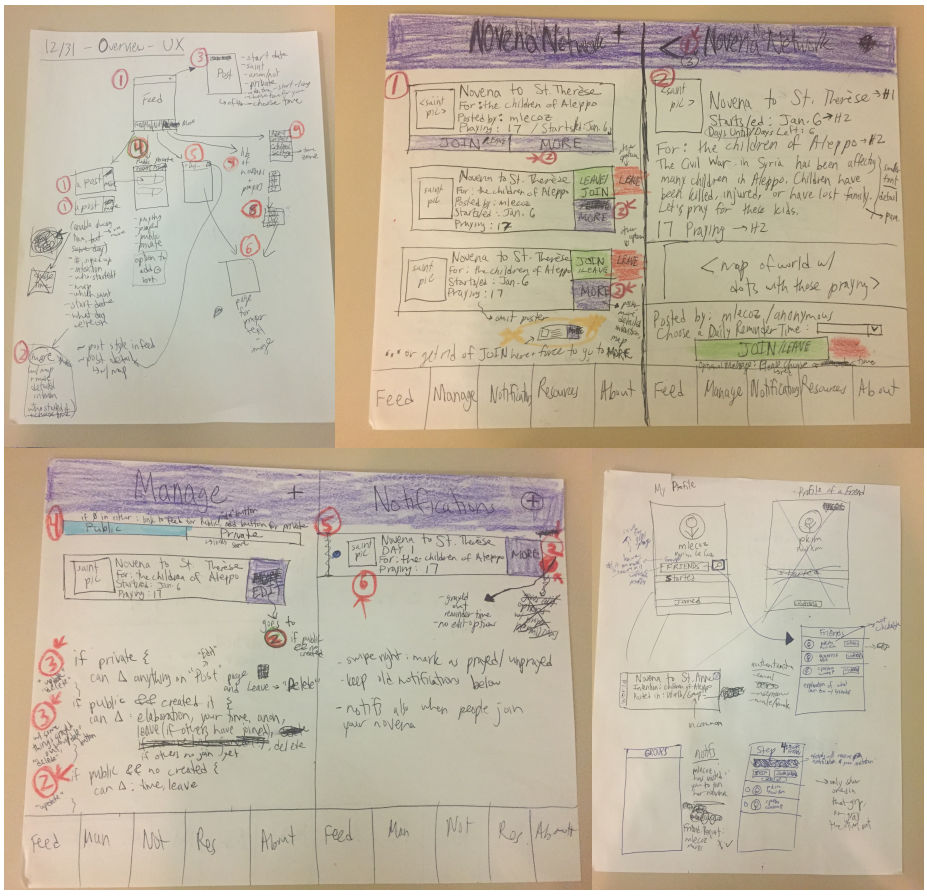

Figure 12. Some sketches from December 31, 2016

After this sketching, I made a series of three new prototypes created with JustInMind. After creating each prototype, I sought feedback from friends whom I perceived as potential users of the app. The remainder of this section will examine the major elements of these three designs and how my designs led to the actual version of NovenaNetwork I implemented.

c. Design Iteration 1

If I had to assign my first design a superlative, it would be "most confusingly organized." Of course, upon completing it, I thought it was brilliant, which is usually how people feel about their own designs. Upon talking to my friends and watching them interact with my prototype, I soon discovered that what I had created was a mess, at least organizationally. My five tabs were "Feed," "Manage," "Notifications," "Library," and "About." In this design, the Feed tab displayed all the novena posts that those on the network had made (like the World tab of my end product), and the Manage tab included all the novena posts that the user had initiated or joined (which is included in the Profile tab of my end product).

The functions of the Feed and Manage tabs spawned great confusion. One friend, Chris Kymn '18, rightly pointed out that "feeds" on social media sites usually include content to which you have ***subscribed***. In my Feed, this was not the case. Another issue, which no one explicitly pointed out, but which I think probably explains some of the confusion, was that the Feed and Manage tabs looked strikingly similar, so it was hard to distinguish their functions—it was not necessarily clear that the former was for novena posts you could join, whereas the latter was for novena posts you had already joined (or initiated). Further, as my friend Paul Kim '17 pointed out, the cog symbol that I used for the Manage tab looked like it was a tab for settings, which was also misleading. He thought that I should rename this tab "My Novenas" and change the symbol, which I did do in my second iteration of this design.

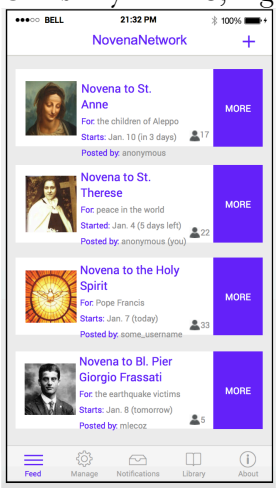

Figure 13. First iteration of the World Feed

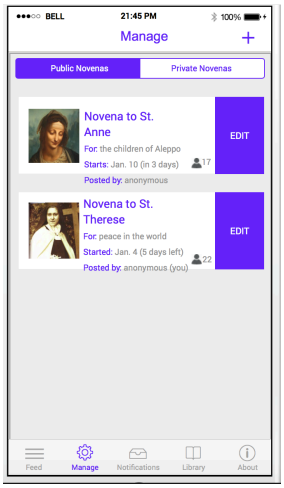

Figure 14. Manage tab from the first design iteration, which was a precursor to the eventual app's Profile tab

Another issue that came up pertained to the Library tab. In this first iteration of my design, the Library tab contained all of the novenas available (this tab was basically identical to Step 1 of the "Start a Novena" process in my end product). During the novena post initiation process, users would have to choose a novena from a list, but if they wanted more information on that novena (such as prayers), they would have to go to the Library tab. My friend Allie Norris '19 made an interesting observation: she said that if she were initiating a novena post, she would not want to have to toggle between a couple different tabs. She said she would want novena information ***as*** she was initiating a novena post. In the next iteration, I moved the contents of the Library tab to

Figure 15. First iteration of the additional details page for a novena post

Step 1 of the novena post initiation process. (Note that in this first iteration, creating a novena post did not even ***have*** steps. That appeared in the second iteration.)

Another confusing feature of this first iteration of my design was the "+" button I used for initiating novena posts. My friends did not realize that this button would allow them to start their own novena post. Hence, they were overall confused about how novena posts on the World Feed even got there. Therefore, not everyone was able to even figure out that NovenaNetwork had a social/community aspect. The "+" button issue was not remedied until my third iteration of the design, when I replaced it with a much clearer "Start a Novena" button.

In spite of this first iteration's many flaws, it did have some redeeming features. For example, the design of the detailed page for novena posts, did not change very much across the three iterations.

d. Design Iteration 2

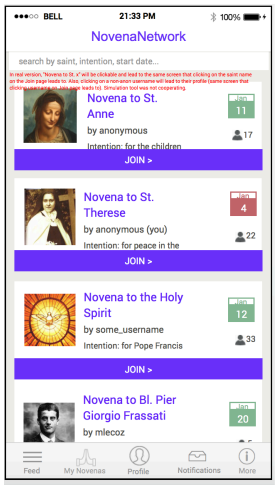

Figure 16. Second iteration of the World Feed

In my second iteration of this design, I tried to remedy the problems that arose in my first iteration, as well as incorporate more aspects of social media. As discussed above, I went overboard with the social media paradigms.

I changed my five tabs to be "Feed," "My Novenas," "Profile," "Notifications," and "About." The My Novenas tab was basically the same as the previous Manage tab, just with a different name and tab symbol. Because I had gotten rid of the Library tab per Allie's feedback on the first iteration, I was able to have a Profile tab. My plan was for user profiles to be publically viewable. Also in this iteration, I gave users the option to include their username on their novena posts (although they would have the option to remain anonymous). Tapping on a username would reveal the user's profile. Profiles displayed the user's followers as well as those s/he was following. I thought that this would be a good way to help people find their friends on NovenaNetwork. I eventually ended up rejecting this idea on the third iteration because a) I do not like the concept of followers/following; unlike friending, it is not mutual, and b) being able to view the number of followers another has could create an unhealthy popularity contest. Again, this app is primarily about praying, not about social media.

In this iteration, I introduced the step-by-step process for initiating a novena post. Step 1 prompted users to select a novena from a library of novenas. Step 2 asked users to specify the visibility of their post. Having the option to add usernames to novena posts made for excessively complicated options here. Users creating a novena post had three options: public post with username, anonymous public post, and private post (which would not be displayed on the feed). Each option took a lot of explaining; when my friends interacted with this prototype, they did not take the time to read everything I had written, and I do not blame them. Step 3 of the novena post creation process remained

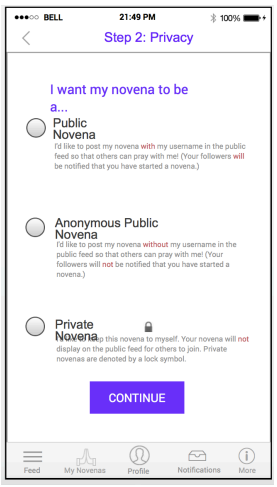

Figure 17. Complicated privacy options in this design

more or less the same after its debut in this iteration. This iteration did not include Step 4 (inviting friends).

This second iteration saw the introduction of a few small touches that persisted through the third iteration. First, it was in this iteration that I introduced a lock icon for private novena posts. (In my third iteration, I gave public novena posts their own icon as well—a globe.) In this second iteration, I altered the post style on the feed to be more consistent with other social media apps. Most social media apps include buttons along the bottom edge of posts ("like," "comment," and "share" buttons in the case of Facebook). In my first iteration, I wanted to be different, so I put a "MORE" button on the right side of the post. In this second iteration, I decided that that was a bit too unconventional. I replaced the "MORE" button with a "JOIN" button at the bottom of each post. (I changed the button to say "JOIN" because I wanted to make it clearer to users that these novena posts were, well, joinable.)

Figure 18. Second iteration of the additional details page for a novena post

In response to my first iteration, my friend Keenan Wood '18 thought that posts, as they were displayed in feeds, were a bit crowded because they had too much text. He thought that a calendar icon might be an interesting alternative to text. I decided to try this in my second iteration. I included a calendar on each post with the start date. If the calendar was green, it meant that the novena post's start date had not yet passed. If the calendar was red, it meant that the novena post's start date ***had*** passed. I was not sure if this was intuitive. When my friends looked at my second prototype and I asked them what they thought the calendar meant, about half understood immediately, while the other half were very confused. I decided to scrap the colored calendar idea but keep a gray calendar for iteration three. I eventually threw away the idea altogether when I was creating the actual app; there was no attractive place to put a calendar icon on a post, really. I did, however, still work to make posts aesthetically simpler (by including less text).

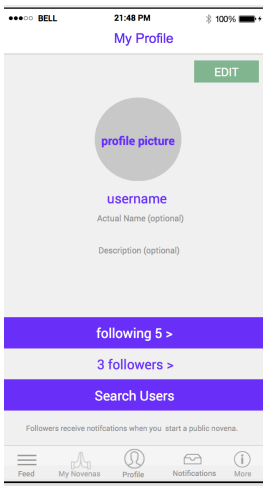

Figure 19. Profile tab from the second design iteration

I made two major changes in iteration three in response to feedback from my friends on this second iteration. My friend Alexa Lewis '18 wondered why the My Novenas and Profile tabs were not combined. The contents of both of these tabs fell under the broad category of "stuff about you;" they logically seemed to go together. I made this change in my third iteration.

The second major change from the second iteration to the third iteration pertained to my use of social media paradigms, as discussed above in "Design Requirements." It was after creating my prototype for this second iteration that I spoke with Meg, the aforementioned campus minister at the Aquinas House, to discuss my concerns about social media paradigms for NovenaNetwork. After this discussion, I decided to make all public posts anonymous

(which consequently simplified the visibility options in Step 2 of the novena initiation process). I also decided to not have publically viewable profiles and to replace "following" with "friending." While I wanted posts to be anonymous, I also did want some way for users to be able to specifically invite their friends to novenas they had created. This is when I thought of Step 4 of the novena post creation process. This step allows users to send a novena invite notification to any friend on their friends list. I find this model to be a good middle ground and an appropriate use of the "friending" feature.

e. Design Iteration 3

In the third iteration of my design, I reorganized my tabs yet again, and this organization persisted to my end product. The five tabs were: "World, "Connect," "Profile," "Notifications," and "More." The World tab was simply a renaming of the Feed tab—nothing functionally changed. I just thought that "World" made it clearer that this page was not a feed of content to which the user had subscribed. Further, I thought it was more powerful for users to know that they are praying with the ***world.*** The Connect tab provided a place to search for friends by username. The Profile tab combined My Novenas and Profile from the second iteration; this tab actually ended up looking almost identical to its analogue in the end product.

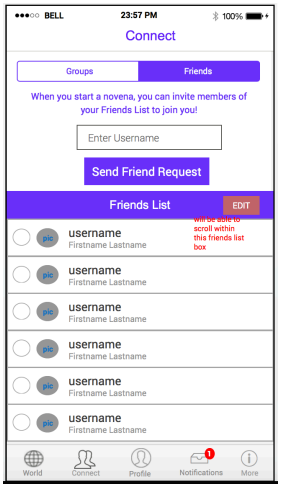

Figure 20. Connect tab from the third design iteration, which strongly resembles the Connect tab in the end product

As discussed in the previous section on the second iteration, in this third iteration I adopted a new model for the social media aspect of NovenaNetwork. I removed the option to display usernames on novena posts (which made the visibility options for a new novena post a lot simpler), and threw out the idea of publically viewable profiles. As a replacement, I introduced "friending" and allowed users to invite specific friends to their novena posts in Step 4 of the novena post initiation process.

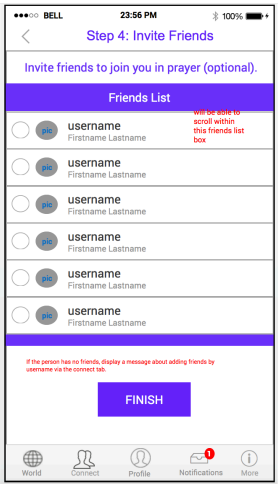

Figure 21. Inviting friends as part of the novena post creation process

In this iteration, I also finally got rid of the "+" button for creating novena posts, which first reared its ugly head in the first iteration. As discussed earlier, my friends who examined my prototypes were unable to predict what the "+" button would do. In this third iteration, I got rid of this ambiguous button, replacing it with a wide "Start a Novena" button anchored at the bottom of both the World tab and the Profile tab. Not only did this change make the button's function clearer, but it also implied that all the novena posts on the World Feed were created by users. Hence, the social aspect of NovenaNetwork was more readily evident.

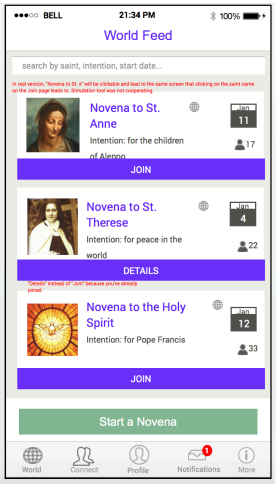

Figure 22. Third iteration of the World Feed

There were several elements of my third iteration that did not make it into my end product. First, in my third iteration I

maintained the calendar icon displaying the novena post's start date from iteration two. It was not until I started designing in Xcode (Apple's IDE) that I realized the calendar was visually clunky. Second, I came up with somewhat of an unconventional design for the Notifications tab in iteration three. I wanted each notification to link to another page, and I thought I had to make it really clear that this was the case. For example, I wanted novena post invite notifications to link to the page for the corresponding novena post. I decided that the best way to do this would be to have an obnoxiously large button (relative to the size of the notification) entitled "View." Once in Xcode, I threw away the button idea and just made the notification itself tappable. Notifications on other social media platforms are tappable, so most users would already know to tap such a notification.

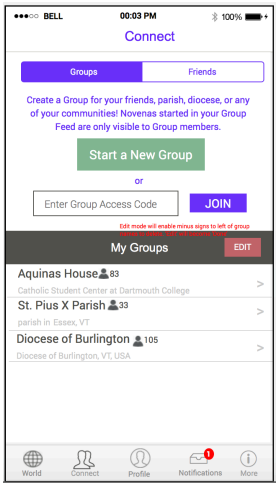

Figure 23. Groups feature from the third design iteration

There were also a couple of ***good*** elements of my third iteration that did ***not*** make it into my end product, but which I would like to add in the future. Most notably, I had designs for a "Groups" feature. "Groups" would allow users to create private feeds for a group of friends, their parish, their diocese, etc. This option would provide a happy medium between private novena posts and world novena posts. This is a feature that I am still very excited to implement, and I anticipate that I will have it ready for release some time this summer.

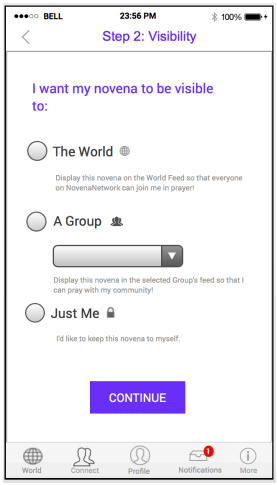

Figure 24. Simplified privacy options in this design

While there were elements of the UI that still did not feel quite right in iteration three (such as notifications and the calendar icon), I finally felt content with the overall flow and organization of the app. I showed this prototype to seven of my friends, and all of them really liked it. From my observing them, they also seemed to intuitively understand how the app worked based on its layout. (And some of the people to whom I showed this prototype had no background on the project and had not seen the two previous prototypes.) The priest at the Aquinas House, Father Brendan Murphy, summed it up nicely when he simply said, "I get it."

f. Logo Design

I did not design my logo until I was almost done with coding NovenaNetwork, but since this is the design section of the paper, I will briefly mention a bit about how the logo came to be. À la COSC 67, I brainstormed by sketching. I eventually settled upon a design that is an "N" for NovenaNetwork, while simultaneously incorporating into the letter a Christian fish symbol called the ***ichthus***. In the earliest days of the Church,

Figure 25. Final logo

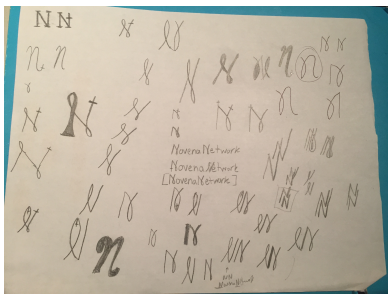

Figure 26. Logo brainstorming

when being a Christian in the Roman Empire was extremely dangerous, Christians used the ichthus as a secret symbol to indicate to each other that they were fellow Christians. Note the vertically oriented (nose down) fish in the loop of the "N."

5. Implementation

Prior to coding NovenaNetwork, I had no experience programming for iOS. I did not know how to use Xcode, nor did I know the Swift programming language. In an internship my junior winter, I did a little bit of Android programming, but I never built anything from scratch and did not really understand how everything fit together. To learn coding for iOS, I watched most of the YouTube videos from Stanford's iOS 9 course.[10] (I was actually programming for iOS 10, but I was able to apply knowledge of iOS 9 to coding for the subsequent operating system. Stanford has not yet released videos for an iOS 10 course.) To learn Swift 3, I read a lot of the online Swift manual provided by Apple. Although Swift apparently has some performance issues according to some programmers, it has become my favorite language because of how syntactically clean it is.

NovenaNetwork consists of approximately 8,184 lines of code across 37 files. This does not include code generated by Xcode's interface builder system. I began my implementation by building the UI for the most essential parts of the system (displaying the World Feed, joining and leaving novena posts, creating novena posts, and displaying joined/initiated novena posts in the user's profile). In these early phases, I did nothing with the Connect tab or the Notifications tab, and I limited the four-step novena post creation process to its very basics. I was not yet ready to make a decision about what sort of cloud database service to use, so I made dummy data. I made my first code-related commit on GitHub on January 21, 2017 (having spent the first couple weeks of the term iterating on designs). On May 22, 2017 I made my last code-related commit (for the version currently in the App Store) after 758 commits on GitHub. Having spent so much of COSC 99 coding, I could write a book on my implementation. Because this is not a book, I will spend the remainder of this section discussing a) how I structured my database, b) three interesting technical challenges I encountered and how I addressed them, and c) how the skills I have acquired as a Computer Science major enabled me to complete this project.

a. Data Storage

Originally, I thought that I would store NovenaNetwork's data using a well-known service called Heroku. After examining what Heroku provides as part of its free tier, I was not sure that it would suit my needs. In the Stanford videos, I learned about CloudKit, which is Apple's own cloud storage system available for anyone with an Apple Developer Account. Anyone who wants to submit an app to the App Store must have such an account, which costs $99. I would inevitably have to pay this fee eventually, so I went ahead and got my account and used CloudKit. CloudKit provides 10 GB of asset storage, 100 MB of database storage, 2 GB of data transfer, and 40 requests per second.[11] All of this scales with the number of users the app acquires. CloudKit also makes authentication easy: users do not need a password to log into a CloudKit app—instead, they merely have to be logged into their iCloud account on their device.

CloudKit does not provide an intricate backend. It is merely a database. Developers can create ***record types*** with ***fields*** (either programmatically or through the CloudKit dashboard). ***Records***, or instantiations of record types, are stored in ***zones***. Every app is provided with a public ***default zone***, which any user's account can access. Each user also has a private default zone, which only his/her account can access. NovenaNetwork only uses the public default zone because using private default zones eats into individual users' iCloud storage. (***Shared zones*** can also be created for data shared between a subset of users, but this was not used in NovenaNetwork.) Every app has a built-in record type called ***User***, and ***User Records*** are stored in the public default zone. The developer may add fields to this special User record type. Finally, it is worth noting that each record includes metadata, including the creator, the creation date/time, the modification date/time, etc.

From code, the CloudKit database can be queried by specifying some predicate that defines search guidelines. Individual records can be fetched by their record ID, and there is a special function for getting the record ID of the current user. Records can be created in code and then saved to (or deleted from) the database, either in batches or individually. Because records may relate to each other, records can store ***references*** (or lists of references) to other records. (References are kind of like pointers.) Below, you can see a representation of how I structured my CloudKit database. Each square box represents a record type, and each yellow box lists its fields. Arrows represent references.

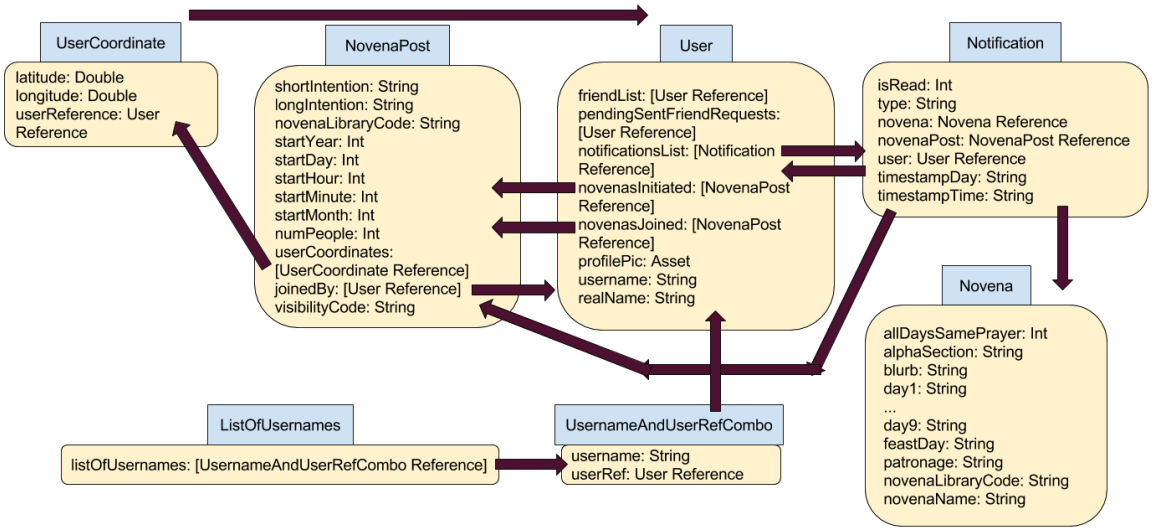

Figure 27. Data storage in the CloudKit database for NovenaNetwork

Here are some notes on less obvious aspects of this organization:

- `NovenaPost`'s field `visibilityCode` is always either `World` or `Private`. This field is used in the query that determines which novena posts will be displayed in the World Feed.
- `User`'s field `pendingSentFriendRequests` includes references to all users to whom the user has sent a friend request (but to which the recipients of the request have not yet responded). This data is used to prevent the user from sending duplicate friend requests.

- `User`'s `novenasInitiated` and `novenasJoined` are used to determine the novena posts displayed on the user's profile. They are also used to determine whether to display "JOIN," "LEAVE," or "DELETE" on a novena post's details page. Further, they are used to determine whether a novena post should read "JOIN" or "DETAILS" in the World Feed.
- `Notification`'s `isRead` is used to determine whether or not the "unread" dot should be displayed on a notification. It is only ever a value of 1 (true) or 0 (false). (CloudKit does not provide `Boolean` types.)
- `Notification` has non-`nil` values for some subset of the fields `novenaPost`, `novena`, and `user`. It depends on which type of notification is being stored. The three types (`type` is another field of `Notification`) are `joined`, `join_me`, and `friend_request`. For example, a `join_me` notification (i.e., a novena invite) would have an associated user (the inviter) and an associated novena post (the novena post to which the notification recipient is being invited).
- Both `NovenaPost` and `Novena` have a field entitled `novenaLibraryCode`. A library code corresponds with each novena offered by the app. For example, Bl. Pier Giorgio Frassati's `novenaLibraryCode` is `BlPierGiorgioFrassati`. Similarly, St. Gianna Beretta Molla's is `StGiannaBerettaMolla`. I noted that images take a ***long*** time to be fetched from the cloud. This is particularly problematic when rendering Step 1 of the novena post creation process since twenty pictures need to be displayed. To solve this problem, I stored all the images for novenas locally. When a `Novena` record is retrieved from the cloud, its `novenaLibraryCode` can be extracted. Then, this code can be used as a key in a Swift dictionary (i.e., map in other languages) to retrieve the appropriate image from the app's local assets.
- Some novenas repeat the same prayer every day, while others have different prayers for each day. `Novena`'s field `allDaysSamePrayer` is 1 if all the days have the same prayer and 0 otherwise. For novenas that have the same prayer every day, this allows me to only store the text of the prayer once, in the field `day1`.
- `ListOfUsernames` and `UsernameAndUserRefCombo` were used to work around a technical limitation of CloudKit and will be discussed later.

b. Technical Challenge #1: Multithreading

Before creating NovenaNetwork, most of my knowledge of multithreading and asynchronicity was purely theoretical (from COSC 10 and COSC 50). I had never really had the opportunity to independently struggle with multithreading. Building NovenaNetwork gave me that opportunity.

Apple products use a technology called "Grand Central Dispatch" (GCD) to optimally assign different tasks to different threads. Developers can have some control over GCD if they choose—they may specify tasks' priorities by putting them in built-in queues of the appropriate priority. For example, a task that the user initiated (by tapping a button, perhaps) might be put in a higher priority queue because the user needs a timely response to his/her action. Some sort of clean-up task that is meant to run in the background, on the other hand, might belong in a lower priority queue.[12]

In the context of my project, multithreading got particularly interesting when I started using CloudKit. CloudKit has a set of "convenience" methods that use a special

Swift feature called a ***closure***. Before I explain how these closures relate to multithreading, we will look at a gutted example from my code that uses such a convenience method.

```
self.db.fetch(withRecordID: novenaPostRecordId) { novenaPostRecord, error in
    if error == nil {
        // do stuff (probably with novenaPostRecord)
    }
    else {
        // handle the error
    }
}
```

The `fetch` method is a convenience method provided by the CloudKit API for retrieving a record in the cloud by its `recordID`. If the above method was successful, it "returns," so to speak, the record as `novenaPostRecord` and a `nil` error. If, on the other hand, something went wrong, `novenaPostRecord` would be `nil` and the error would be an object with important information about what went wrong.

The above syntax likely looks very odd to those who have never seen Swift before. The `fetch` function actually takes ***two*** parameters. The first, naturally, is `withRecordID`. The second is a closure (everything in the outer set of curly braces). Closures are basically another way of writing functions. The closure's parameters, in this case, are `novenaPostRecord` and `error` (which is perhaps hard to wrap your mind around because they are also the return values of `fetch`). The closure is invoked when `fetch` has retrieved the `novenaPostRecord` and the `error`. The closure is basically like a completion handler.

So where does multithreading come into all of this? As it turns out, CloudKit's closures generally run on different threads. This was not evident to me when I first started using CloudKit. This had two practical ramifications that initially prompted a lot of confusion.

First, since I had never really programmed with multithreading in mind, I would try to do things like this:

```
self.container.fetchUserRecordID() { userRecordID, error1 in
    if error1 == nil {
        self.userRecordID = userRecordID // save as instance variable
        // do other stuff
    }
    else {
        // handle the error
    }
}

self.db.fetch(withRecordID: self.userRecordID) { userRecord, error2 in
    if error2 == nil {
        // do stuff
    }
    else {
        // handle the error
    }
}
```

From examining this sort of code using Xcode's debug mode, I learned that things were being executed as follows: 1) the `userRecordID` was fetched on the main thread, 2) the associated closure was set aside to execute at some point on a different thread, 3) the main thread would try to execute the `userRecord` fetch, which was sometimes successful and sometimes not. This was because this second fetch depended on the first closure having executed. If the first closure had not executed, then `self.userRecordID` (an

instance variable) was `nil`. This experience taught me that, when working with multiple threads, you ***cannot*** count on tasks executing in the order you coded them (which I theoretically knew, but seeing it in practice was totally different). When using CloudKit's convenience methods, the programmer must instead nest these calls to ensure the proper ordering:

```
self.container.fetchUserRecordID() { userRecordID, error1 in
    if error1 == nil {
        // note that there is no need to store userRecordID as an instance variable now
        self.db.fetch(withRecordID: userRecordID) { userRecord, error2 in
            if error2 == nil {
                // do stuff
            }
            else {
                // handle the error
            }
        }
    }
    else {
        // handle the error
    }
}
```

The second practical ramification of not understanding that CloudKit's closures ran on different threads became evident when I wanted to make UI updates within closures. For example, suppose I coded some sort of UI update (such as displaying the username stored in the newly-retrieved `userRecord`) in the block above where `error2` is `nil`. I simply would not see my UI update take effect (or at least not in a timely manner). As it turns out, CloudKit's closures are not put in the high-priority queues of GCD that are necessary for UI changes. Hence, I learned from the Stanford videos that I needed to "dispatch" the main queue and make any UI updates there, as shown below. This allowed UI updates to display in a timely manner.

```
self.db.fetch(withRecordID: userRecordID) { userRecord, error2 in
    if error2 == nil {
        // do non-UI stuff - on some other thread
        DispatchQueue.main.async(execute: { // change to main thread
            // make UI updates
        })
    }
    else {
        // handle the error
    }
}
```

c. Technical Challenge #2: Querying Usernames

When I wanted to find a record of a particular record type with a certain value for a particular field, I became accustomed to doing something like this:

```
let db = CKContainer.default().publicCloudDatabase

let predicate = NSPredicate(format: "%K = %@", NOVENA_LIBRARY_CODE, self.novenaPost.object(forKey:
NOVENA_LIBRARY_CODE) as! String)

let query = CKQuery(recordType: NOVENA, predicate: predicate)

db.perform(query, inZoneWith: nil) { records, error in
        // ...
}
```

The `predicate` in the above code specifies that I am looking for records that have a certain `novenaLibraryCode` (described above in the notes below the database diagram), and the `query` specifies that I am looking only at `Novena` records.

There were at least two instances in my code where I wanted to query `User` records for a specific `username`. First, if users wish to send friend requests to others, they must type their friend's `username` and tap "Send." I wanted the program to query the `User` records for this `username`. (Recall that `username` is a field of the record type `User`.) Then, the corresponding `User` record could have its `notificationsList` field adjusted to include this friend request. I also wanted to query `User` records by `username` when a first-time user was creating a `username`. I wanted to check to make sure that this `username` had not already been taken.

I soon found out that such a query was not possible. Because the `User` record type is built into CloudKit, it is not queryable in this way.[13] This is merely a technical limitation of CloudKit. To remedy this unfortunate issue, I added two new record types: `UsernameAndUserRefCombo` and `ListOfUsernames`. The former is exactly what it sounds like: records of this type include a reference to a `User` record and the corresponding `username` as a `String`. (Technically, the `User` record contains the `username` already, but as we will see in a bit, it is useful to have easy access to the `username` without having to get the `User` record from the `User` reference, which requires a `fetch` operation.) The `ListOfUsernames` record type has but one field: a list of `UsernameAndUserRefCombo` references. The `ListOfUsernames` record type is only instantiated once, such that there is one list representing all users of NovenaNetwork.

When a user opens NovenaNetwork, the system checks whether his/her `User` record has a `username`. If it is `nil` (which would be the case for a first-time user), the user is greeted by a create-a-username screen. Upon submission of the desired `username`, the system searches the `ListOfUsernames` to see whether that `username` has been taken. If it has been taken, the user is told to choose a different `username`. If it has not been taken, a `UsernameAndUserRefCombo` is instantiated and added to the `ListOfUsernames`. When the user subsequently opens NovenaNetwork, s/he will be taken straight to the World Feed screen because s/he has a non-`nil` username. This ensures that each user is only added to the `ListOfUsernames` once.

When users try to send a friend request to a user with some `username`, `ListOfUsernames` is searched for that `username`. If it exists, the corresponding `User` record is fetched (from the `User` reference in `UsernameAndUserRefCombo`), and a friend request notification is added to the requestee's `notificationsList`.

It is worth noting that all searches of the `ListOfUsernames` are currently linear and therefore run in O(n) time, where n is the number of users on NovenaNetwork. This will be a problem as the number of users on NovenaNetwork grows. Part of my motivation for including the `username` as a `String` in `UsernameAndUserRefCombo` is that this will make it easier to keep the `ListOfUsernames` in sorted order; that way, I can eventually implement a binary search instead of a linear search, thus reducing my runtime to O(log(n)).

d. Technical Challenge #3: The Agony of Error Handling with CloudKit

As shown in the code in Technical Challenge #1, CloudKit operations can return errors. I did not bother handling and responding to these errors until the very end of my coding process. I found that I had included approximately 100 CloudKit operations in

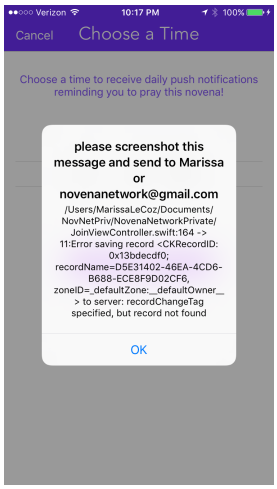

Figure 28. Error message for beta testers

my code, and each could result in an error. Further, there were 33 different error possibilities for each operation.[14] This did not even account for "partial errors," which occur, for example, when only ***certain*** records in a batch do not properly save to the database. Another issue was that the names of CloudKit errors were not always helpful for figuring out what they meant. (For example, `CKErrorUnknownItem` meant that the record the user tried to `fetch` could not be found in the database; by its name, I would have thought that this error was for problems that did not fall under any other error category.) To figure out what sorts of errors occurred most frequently, I incorporated pop-up error messages in my beta testing version that testers could screenshot and send to me, seen at left. (I removed these messages for my final version of the app, of course.)

My initial approach to error handling was to create a function containing a large switch statement with 33 switches, one for each error. Then, I would call this function whenever the error from a CloudKit operation was non-`nil`. This worked all right for certain types of errors that can be generically handled. For example, the errors `CKErrorNetworkUnavailable` and `CKErrorNetworkFailure` occur when the Internet connection is either unavailable or poor, respectively. Generally, I can respond to these errors by providing a pop-up message telling users that their Internet connection is bad. (There were a couple of special instances, though, that needed to be specially handled. For example, if the Internet fails on NovenaNetwork's initial loading screen, I display a pop-up message ***and*** provide a "Try Again" button for the user to tap. While users can easily "try again" on a screen like the World Feed by pulling down to refresh, there is no way to "try again" on the loading screen without a button.)

All in all, the switch statement function worked well for some errors, but I soon discovered that there were several types of errors that needed to be handled in their own very unique contexts. The most prevalent issue of this nature occurred when multiple users would try to edit the same record at (approximately) the same time. We will consider a generic example of this scenario.

Suppose that Suzy and Bob both try to join a novena post at approximately the same time. Both Suzy and Bob's tapping the "Done" button invokes a block of code that edits the original novena post record (augmenting the number of users praying and adding user coordinates to the `userCoordinates` list). We will call this original, pre-edited novena post record $\boldsymbol{NPR_{ancestor}}$. Suppose that Suzy's block of code finishes before Bob's does. Suzy's edited novena post record, $\boldsymbol{NPR_{client\_suzy}}$, is saved to the database, replacing $\boldsymbol{NPR_{ancestor}}$. When the block of code tries to save Bob's novena post record, $\boldsymbol{NPR_{client\_bob}}$, to the database, he gets an error called `CKErrorServerRecordChanged`. This indicates that the record on the server has been changed since Bob started editing it. Hence, $\boldsymbol{NPR_{client\_bob}}$ is "behind." This error is sort of like a GitHub merge conflict.

The error `CKErrorServerRecordChanged` is ***supposed to*** provide the programmer with three records to help resolve this conflict: the record the user attempted to save (client record), the record the user had been editing (ancestor record), and the version of this record currently on the server (server record). The programmer is supposed to modify the server record (using information from the client and ancestor records to help) and re-save it to the database. When I was trying to extract fields that were of type `List` from the

ancestor record, I kept getting crashes because these fields were apparently `nil`. I spent hours upon hours trying to figure out what was going wrong (this was the worst bug of the project). I eventually printed out each of the three records: client, ancestor, and server. What I found was that all `List` fields (such as the reference list called `userCoordinates`) were completely omitted from the ancestor record. These lists were not merely empty; rather, it was as though the ancestor record had no such fields. The server and client records, however, did have these fields. This seems to be a CloudKit bug. Others on StackOverflow have had a similar problem.[15] Once I knew that the ancestor record was unreliable, I simply stopped trying to use it and found other ways to get the information I needed.

As I examined my code, I found that this error, `CKErrorServerRecordChanged`, could occur in ***many*** places. Records of type `ListOfUsernames`, `NovenaPost`, and `User` could all be written to by multiple users. Here are a few scenarios that could result in a `CKErrorServerRecordChanged`: 1) two users try to join NovenaNetwork at approximately the same time, 2) two users try to leave a novena post at approximately the same time, and 3) one user, ***A***, deletes a friend from his/her friends list while another user accepts a friend request from ***A*** (both of these actions result in editing ***A***'s friends list). There were about 35 scenarios like this, and each had to be handled individually because the conditions of each were so unique. Below, you will see the function invoked when two users try to join NovenaNetwork at approximately the same time and a `CKErrorServerRecordChanged` error occurs:

```
func tryAgainOnAddingToUsernameList(err4: CKError) {

    let serverRecord = err4.userInfo[CKRecordChangedErrorServerRecordKey] as! CKRecord
    let recordTriedToSave = err4.userInfo[CKRecordChangedErrorClientRecordKey] as! CKRecord
    // note that I don't bother trying to get the ancestor record

    var listOnServer = serverRecord.object(forKey: LIST_OF_USERNAMES) as! [CKReference]

    let listUserTriedToSave = recordTriedToSave.object(forKey: LIST_OF_USERNAMES) as! [CKReference]
    let oldListCount = listUserTriedToSave.count
    let whatUserTriedToAppend = listUserTriedToSave[oldListCount - 1] as CKReference

    listOnServer.append(whatUserTriedToAppend)

    serverRecord[LIST_OF_USERNAMES] = listOnServer as CKRecordValue?

    self.db.save(serverRecord) { record, err in // try to save again
        if err == nil {
            DispatchQueue.main.async(execute: { // UI updates
                self.working.stopAnimating()
                self.working.isHidden = true
                // go to the main page
                let vc = UIStoryboard(name: "Main", bundle:
nil).instantiateViewController(withIdentifier: REGULAR_START) as! UITabBarController
                self.present(vc, animated: false, completion: nil)
            })
        }
        else {
            if (err as! CKError).code.rawValue == 14 { // if CKErrorServerRecordChanged again
                // recursively call this function again
                self.tryAgainOnAddingToUsernameList(err4: err as! CKError)
            }
            else { // some other error => call the switch statement
                ErrorHandlers.handleCKError(err as! CKError, self.working, self, #file, #line)
            }
        }
    }
}
```

Suppose that this function is invoked because there was a conflict. Now suppose that, while this second attempt to join NovenaNetwork is being made, yet ***another*** new user beats you to the punch. This function can handle that because it recursively calls itself if a `CKErrorServerRecordChanged` occurs again. Further, if another type of error occurs, my generic error handling switch statement is invoked. This is just one example of error handling in NovenaNetwork. Many other edge cases were addressed, pertaining to both `CKErrorServerRecordChanged` and other errors.

e. Culminating Aspects of this Project

Designing and developing NovenaNetwork drew on many topics covered in my Computer Science coursework and internships, including:

- Object-oriented programming – Swift is an object-oriented language, and iOS is laden with object-oriented paradigms like inheritance, polymorphism, and interfaces (delegates in Swift). For example, nearly all of the views I created were subclasses of iOS's built-in UIViewController. I also made use of custom classes in my code; for example, during the four-step novena post creation process, I store the user's inputted data across these fours steps in an object of type `NewNovenaPost`, which has properties such as `shortIntention`, `longIntention`, `novenaLibraryCode`, etc. This object is passed between `ViewControllers` until it is time to save data to the cloud at the end of the creation process.
- Recursion – As discussed in Technical Challenge #3, I used recursion to address the possibility of more errors occurring as a result of error handling.
- Multithreading – As discussed in Technical Challenge #1, I dealt with multithreading when using CloudKit.
- Design Patterns – iOS uses the Model-View-Controller (MVC) design pattern, where the Controller communicates with the Model (such as the database) and the View (the UI) separately, and the Model and View do not communicate with each other. This makes for nice abstraction.
- Data Structures – Lists and Dictionaries. Dictionaries, for example, were used to expedite the retrieval of information about novenas (images, etc.).
- Database Conflicts – I had to think about what would happen if multiple users tried to write to the same record at the same time, as discussed in Technical Challenge #3.
- Abstraction – I wrote clean code by using object-oriented paradigms and functions to cultivate the appropriate level of abstraction.
- Memory Management – Swift is not terribly prone to memory leaks, but there are a few circumstances where they occur, and I had to address these issues.
- Git – I used the command-line GitHub tool for version control.
- Error Prevention – I had to anticipate weird things the user might try to do and prevent those things from happening. As just a couple of examples, I do not allow the user to ***not*** enter a prayer intention when creating a novena, and I do not allow the user to choose a novena start date in the past.
- Testing – I was very meticulous in my testing of features as I added them. This was something we were taught to do in COSC 50. I tested ***all*** the scenarios I

thought I had coded for to make sure they all worked. Although I did not formally document my tests, I basically carried out unit testing.

- Shell Scripting – I wrote a bash script to back up my work every hour in case I forgot to manually do so.
- Designing a Software System – The "Design" in COSC 50: Software Design and Implementation. I had to figure out what sort of data to store to make the system as efficient as possible.
- Design Process – I underwent the design process covered in COSC 67: Investigation, Ideation, Prototyping, Evaluation, and Production. Further, although I was coding in my junior winter internship, I worked in an office full of designers, so I would often sit in on design meetings or overhear conversations that helped me learn the kinds of things designers think about. Additionally, I also learned about design during my time working for the DALI Lab although I was formally a developer.
- Design Heuristics – Minded design heuristics from COSC 67:
    - Visibility of System Status – For example, I provided users with an activity indicator ("spinny" wheel) to let them know when the system is doing something behind the scenes.
    - Match between System and Real World – For example, my buttons use the average person's language ("JOIN," "DONE," etc.), not the developer's language.
    - User Control & Freedom – For example, I always provide users with a Cancel or back button when they go to a new page.
    - Consistency and Standards – I use iOS paradigms such as, for example, positioning an Edit button (on the Profile page) in the upper right hand corner of the interface.
    - Help users recover from errors – For example, I provide users with error messages that suggest what they should do to resolve the error. As another example, I provide a "Try Again" button (discussed earlier) when an error occurs on the "loading" screen.
    - Error Prevention – For example, I include code to prevent users from doing weird things like choosing a novena post start date that is in the past.
    - Recognition rather than Recall – For example, when users choose a novena in Step 1 of the novena post creation process, they can view information about these novenas from a button on that screen. They do not need to go to a library in a different tab to read about the novenas they are considering (as was the case in some of my earlier designs).
    - Aesthetics and Minimalist Design – My interface is not visually crowded or overwhelming.
    - Help and Documentation – My About page describes the purpose of the app and provides a contact email to which questions and concerns may be sent.
- Cloud Computing – I was on a cloud computing team for my junior summer internship.
- DeMorgan's Law – I always find DeMorgan's law helpful when I am coding semi-complicated '`if`'-'`else if`'-'`else`' ladders.

- Using APIs – I used the CloudKit API. I have used APIs very frequently in internships, as well as in classes (although less frequently).
- Making use of the open-source community – I used one open-source library in NovenaNetwork (for the radio buttons in Step 2 of the novena post creation process). I have very often used open-source libraries in internships.
- Coding in general – I had never coded before taking COSC 1 my freshman spring.

6. Evaluation

I released my first beta version of NovenaNetwork to a group of testers in the last few days of April 2017. In total, I had 23 beta testers, many of whom I knew personally, but several of whom were friends of friends. During beta testing, I worked on error handling and push notifications. My main goal with beta testing was to start getting feedback on the app and to identify the most common errors that occurred for users so that those could be addressed before submitting to the App Store.

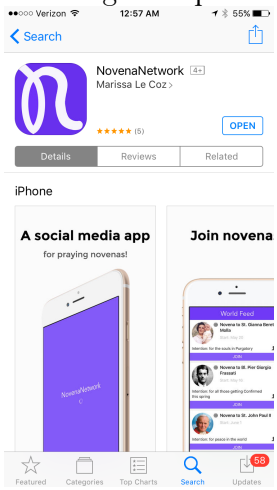

Figure 29. NovenaNetwork in the App Store

On May 16, 2017, I submitted to the App Store. A couple days later, I was approved, but I chose to not release that version because I had some questions about copyrights on the images I had used in the app. After speaking to Dartmouth librarian Barbara DeFelice, who knows a lot about copyrights, I changed some of my images in the app and resubmitted to the App Store. On May 23, 2017, NovenaNetwork entered the App Store. On May 29, 2017, NovenaNetwork had been downloaded 52 times, including once in Russia and twice in China!

What follows is some of the feedback I have received with regard to NovenaNetwork, both for the beta version and for the App Store version.

a. Positive Feedback

"I think this is the app the world needs right now. I am totally serious. There's nothing like this that connects a person with an age-old faith with the elegance of streamlined, modern technology. It's only now that I realize how amazing and world-changing this has the potential to be."
- Xanthe Kraft '16, in person

"I really like the logo and color theme! It's very aesthetic and clever."
- Kevin Patterson '17, via email

"I wouldn't say I'm Catholic, but I definitely need this app!"
- '17 who prefers to remain anonymous, via email

"I really like the app."
- Father Raymund Snyder, O.P., via email

Figure 30. Father Raymund tweeted about NovenaNetwork

"Your app looks great." and "Re: St Joseph: nice layout. Like the pin showing who is praying it; like the pictures."
- Dr. James Filiano, doctor at DHMC, via text message

"Great app! Love being able to connect with other Catholics! Very straightforward and user-friendly.
- MKCahill, App Store review

"I love this app! There are so many wonderful novenas at my fingertips, which the app kindly reminds me to pray. It is also so encouraging to see those praying beside me!"
- Emskillet31, App Store review

"This app is a joy to use. It is very easy to navigate and very satisfying!"
- VTCatholic, App Store review

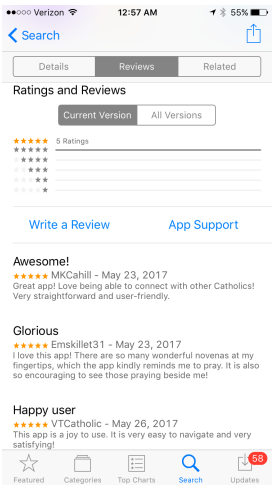

Figure 31. App Store ratings and comments so far

b. Constructive Feedback

While users seem to like NovenaNetwork on the whole, they did offer several suggestions for improvement. Dr. Filiano, quoted above, wished he could edit his prayer intentions after he had posted them. I had toyed with adding this feature during the development process but had decided not to because I was afraid that users might totally change the gist of their prayer intention after others had already signed up to join. After thinking about it more, I would like to add this feature because the creator of the novena post should have the freedom to make edits; if s/he were to change the intention drastically, users who had joined could always choose to not participate.

My friend Marcos Robertson-Lavalle '17 wanted to be able to join novena posts whose start date had already passed. This was actually my original plan in my design phase, but then at the last minute I was not sure if it made sense, so I did not allow for late joins. In retrospect, I think this was a terrible idea; version 1.1 will definitely let people join novena posts late.

My friend Paul Kim '17 wanted a better way to find friends on the app. He did not want to have to ask his friends for their usernames in real life in order to send them friend requests. One way of making friends easier to find would be by allowing for some sort of syncing of the user's contacts with NovenaNetwork. Another solution could involve using Facebook's API to find friends.

A couple of my friends wished that they could see which users had joined a novena post. They also wished that they could receive a notification if a friend whom they had invited to a novena post accepted their invitation. Instinctively, I had these same feelings when using NovenaNetwork; most social media enable these sorts of interactions. As discussed in the design section above, however, the choice to not enable these sorts of interactions was intentional, and I still believe that this was the right decision.

7. Conclusion & Future Plans

The enthusiasm that NovenaNetwork has already generated in the short time it has been available to others confirms that there is a robust market for this product. Designing and developing NovenaNetwork has been a true joy, not only because I was able to work on something that I care so much about, but also because I was able to see much of the Computer Science I have learned as an undergraduate come together in this project. What would have been an unthinkable undertaking three years ago has been made possible through the knowledge and problem solving skills I have gained as a Computer Science major. Sharing my work with the world via the App Store has been very exciting, and I want to keep improving the app for my present and future users. Early this summer, I would like to roll out version 1.1, which will include the following improvements, ranked in order of importance:

1) Allow users to join novena posts late.
2) Allow users to edit their prayer intentions.
3) Fix a UI bug a user discovered on iPad. (The "Continue" button does not fit fully on the screen in Step 2 of the novena post creation process.)
4) Display a pop-up to users asking them to rate NovenaNetwork in the App Store; this is how apps move up in App Store search results.
5) For each novena post, display what day it is on (if applicable).
6) Add a few more novenas to the collection.
7) Add a red badge icon to the Notifications tab symbol, identifying how many unread notifications the user has.
8) Fix a bug affecting the alphabetization of the friends list.

My longer-term plans include adding some more significant features, such as Groups (discussed in the design section) and a way to more easily find friends on the app. I look forward to seeing what the future holds for NovenaNetwork.

8. Acknowledgements

I would like to thank the following people for looking at my design prototypes:

Allie Norris '19, Keenan Wood '18, Margaret Cross '19, Paul Kim '17, Father Brendan Murphy, O.P., Chris Kymn '18, Father Raymund Snyder, O.P., Miles Temel '20, Alexa Lewis '18, Meg Costantini, Ben Palmer '15, Caroline Petro '18, Emmaline Berg '13, Elle Nutter 'GR, Jess Heine '19, and Chris Le Coz (my dad).

I would like to thank all of my beta testers:

Jeff Poomkudy '20, Bridget Shaia '15, Hailey Scherer '20, Xanthe Kraft '16, Xanthe's mom, Xanthe's sister Yvette Kraft, Maura Cahill '20, Emily Dryden '19, Jason Qian '19, Alexa Lewis '18, Dr. James Filiano, Paul Kim '17, Meg Costantini, Ellen Weburg '14, Winfield Tan (friend of Ellen), another friend of Ellen whose name I do not know, Nat Mendolia '19, Marie Le Coz (my mom), Mackenzie Carlson '17, Father Raymund Snyder, O.P., Professor Xing-Dong Yang, Kevin Patterson '17, and Marcos Robertson-Lavalle '17.

Thank you also to my advisors Professor Xing-Dong Yang and Professor Sean Smith, and thank you to Professor Paul Hegarty of Stanford, whose lecture videos I used to learn iOS. Finally, as all developers must say, thank you, StackOverflow.

9. References

[1] Fr. William Sunders, "What is a novena?," ***Catholic Straight Answers,*** accessed 25 May 2017, <http://catholicstraightanswers.com/what-is-a-novena/>.
[2] Acts 1:4 (New Revised Standard Version).
[3] Acts 1:14 (NRSV).
[4] Father Ryan Erlenbush, "What is the significance of a novena? Why nine days?," ***The New Theological Movement***, 18 May 2012, accessed 25 May 2017, <http://newtheologicalmovement.blogspot.com/2012/05/what-is-significance-of-novena-why-nine.html>.
[5] Acts 2:4 (NRSV).
[6] Fr. William Sunders, "What is the role of novenas today?," ***EWTN,*** 25 August 1994, accessed 25 May 2017, <https://www.ewtn.com/library/ANSWERS/NOVENA.HTM>.
[7] Eric Lyman, "Latin Mass resurgent 50 years after Vatican II," ***USA Today,*** 12 March 2015, accessed 25 May 2017, <https://www.usatoday.com/story/news/world/2015/03/12/catholicism-latin-mass-resurgence/70214976/>.
[8] "Declining Number of Catholics," ***Pew Research Center,*** 11 May 2015, accessed 25 May 2017, <http://www.pewforum.org/2015/05/12/americas-changing-religious-landscape/pf_15-05-05_rls2_catholic200px-1/>.
[9] Matthew 6:6 (NRSV).
[10] Paul Hegarty, ***Stanford iOS***, YouTube channel, <https://www.youtube.com/channel/UCYVp6suz7ztKAKY8jpfACXA>.
[11] "iCloud for Developers," ***Apple,*** accessed 26 May 2017, <https://developer.apple.com/icloud/>.
[12] Stanford iOS, "Lecture 8. Multithreading and Text Field," YouTube video, 14 May 2016, < https://www.youtube.com/watch?v=_ib-m6ZCyyo>.

[13] Jonathan, "Querying CloudKit Users Record gives 'Can't query system types,'" ***StackOverflow, 16 December 2014, accessed 28 May 2017,*** **<**https://stackoverflow.com/questions/27510024/querying-cloudkit-users-record-gives-cant-query-system-types>.

[14] "CKErrorCode," ***Apple API Reference,*** accessed 28 May 2017, <https://developer.apple.com/reference/cloudkit/ckerrorcode>.

[15] Coder1224, "Error.userInfo[CKRecordChangedErrorAncestorRecordKey] has no values when CKErrorCode.ServerRecordChanged is reported," ***StackOverflow***, 17 November 2016, accessed 28 May 2017, <https://stackoverflow.com/questions/40645663/error-userinfockrecordchangederrorancestorrecordkey-has-no-values-when-ckerror>.